FDA

```
###-----------------------------------------------------------------------------------------
###-----------------------------------------------------------------------------------------
### Functional Data Analysis
###-----------------------------------------------------------------------------------------
###-----------------------------------------------------------------------------------------

library(fda)
library(fields)
library(fdakma)

###-----------------------------------------------------------------------------------------
### Part I: Smoothing
###-----------------------------------------------------------------------------------------
#         ------------------------------------------------------
#         |         | location_1 | location_2 | .. | location_n |
#         ------------------------------------------------------
#         | time_1 |     ..     |     ..     | .. |     ..     |
#         | time_2 |     ..     |     ..     | .. |     ..     |
#         |   ..   |     ..     |     ..     | .. |     ..     |
# If it's not like that (but transposed):
#    data = t(data)

data = CanadianWeather$dailyAv[,,1]
head(data)
dim(data)
n=dim(data)[2]

matplot(data, type='l', xlab='time', ylab='feat')

# --------------------------------
# Fourier basis (periodic)
time         = 1:dim(data)[1]
nbasis       = 109
basis.1      = create.fourier.basis(rangeval = c(0,dim(data)[1]), nbasis = nbasis)
plot(basis.1)
data.fd.1    = Data2fd(y = data, argvals = time, basisobj = basis.1)
plot.fd(data.fd.1)

# "Report the first 3 coefficients of Station1"
as.numeric(data.fd.1$coefs[1:3, 'Station1'])
# --------------------------------

# --------------------------------
# Least squares basis (without penalization)
time         = 1:dim(data)[1]
m            = 5                          # spline order
degree       = m-1                        # spline degree
nbasis       = 9                          # number of basis
basis.2      = create.bspline.basis(rangeval = c(0,dim(data)[1]), nbasis = nbasis)
plot(basis.2)
data.fd.2    = Data2fd(y = data, argvals = time, basisobj = basis.2)
plot.fd(data.fd.2)
# --------------------------------

# --------------------------------
# Least squares basis (with penalization)
time           = 1:dim(data)[1]
m              = 5                          # spline order
degree         = m-1                        # spline degree
NT             = length(time)
breaks         = time[((0:floor(NT/2))*2)+1]
basis.3        = create.bspline.basis(breaks, norder = m)
plot(basis.3)
data.fd.3      = Data2fd(y = data, argvals = time, basisobj = basis.3)

# For one cruve
#  functionalPar = fdPar(fdobj=basis.3, Lfdobj=3, lambda=1e-8)
#  Xss           = smooth.basis(time, data[,1], functionalPar)
#  Xss0          = eval.fd(time, Xss$fd, Lfd=0)
#  Xss1          = eval.fd(time, Xss$fd, Lfd=1)
#  Xss2          = eval.fd(time, Xss$fd, Lfd=2)
#  df            = Xss$df   #  the degrees of freedom in the smoothing curve
#  df
#  gcv           = Xss$gcv  #  the value of the gcv statistic
#  gcv
#  NT         = dim(data)[2]
#  rappincX1 = (data[3:NT,1]-data[1:(NT-2),1])/(time[3:NT]-time[1:(NT-2)])
#  rappincX2 = ((data[3:NT,1]-data[2:(NT-1),1])/(time[3:NT]-time[2:(NT-1)])-
#               (data[2:(NT-1)]-data[1:(NT-2)])/(time[2:(NT-1)]-time[1:(NT-2)]))*
#               2/(time[3:(NT)]-time[1:(NT-2)])
#  plot(time, data[,1], xlab="t", ylab="observed data")
#  points(time, Xss0, type="l", col="blue", lwd=2)
#  plot(time[2:(NT-1)], rappincX1, xlab="t", ylab="first differences x",type="l")
#  points(time,Xss1 ,type="l",col="blue",lwd=2)
#  plot(time[2:(NT-1)],rappincX2,xlab="t",ylab="second differences x",type="l")
#  points(time,Xss2 ,type="l",col="blue",lwd=2)
```

```r
# -------------------------------

# -------------------------------
# -------------------------------
# For one curve
#    curve: data[,1]
#    basis: basis.1

# Smooth curve
basismat0 = eval.basis(time, basis.1)
Xsp0      = basismat0 %*% lsfit(basismat0, data[,1], intercept=F)$coef
plot(time, data[,1])
points(time, Xsp0, type='l', col='blue', lwd=2)

# First derivative
# finite differences
NT        = dim(data)[2]
rappincX1 = (data[3:NT,1]-data[1:(NT-2),1])/(time[3:NT]-time[1:(NT-2)])
basismat1 = eval.basis(time, basis.1, Lfdobj=1)
Xsp1      = basismat1 %*% lsfit(basismat0, data[,1], intercept=F)$coef
plot(time[2:(NT-1)], rappincX1, xlab='t', ylab='first derivative', type='l')
points(time, Xsp1, type='l', col='orange', lwd=3)

# Second derivative
# finite differences
rappincX2 = ((data[3:NT,1]-data[2:(NT-1),1])/(time[3:NT]-time[2:(NT-1)])-
                (data[2:(NT-1)]-data[1:(NT-2)])/(time[2:(NT-1)]-time[1:(NT-2)]))*
             2/(time[3:(NT)]-time[1:(NT-2)])
basismat2 = eval.basis(time, basis.1, Lfdobj=2)
Xsp2      = basismat2 %*% lsfit(basismat0, data[,1], intercept=F)$coef
plot(time[2:(NT-1)], rappincX2, xlab="t", ylab="second derivative", type="l")
points(time, Xsp2, type='l', col="orange", lwd=3)

# Approximate pointwise confidence intervals (one at the time!)
# we can estimate the variance of x(t) as: sigma^2*diag[phi*(phi'phi)^{-1}(phi)']
S        = basismat0 %*% solve(t(basismat0) %*% basismat0) %*% t(basismat0) #projector
sigmahat = sqrt(sum((data[,1]-data[,1])^2)/(NT-nbasis))        #estimate of sigma
lb       = Xsp0 - qnorm(0.975) * sigmahat * sqrt(diag(S))
ub       = Xsp0 + qnorm(0.975) * sigmahat * sqrt(diag(S))
plot(  time, Xsp0, type="l", col="blue", lwd=2, ylab="")
points(time, lb,   type="l", col="red",  lwd=2, lty="dashed",)
points(time, ub,   type="l", col="red",  lwd=2, lty="dashed")

# -------------------------------
# Choosing the number of basis by generalized cross-validation
#   basis: bspline (m=5)
nbasis = 6:150
gcv    = numeric(length(nbasis))
for (i in 1:length(nbasis)){
  basis  = create.bspline.basis(c(0,dim(data)[1]), nbasis[i], m)
  gcv[i] = smooth.basis(time, data[,1], basis)$gcv
}
plot(nbasis, gcv)
nbasis[which.min(gcv)]
# -------------------------------
# -------------------------------

###----------------------------------------------------------------------------------
### Part II: Mean and Covariance
###----------------------------------------------------------------------------------
# Mean
plot.fd(data.fd.1)
lines(mean.fd(data.fd.1), lwd=3)

plot.fd(data.fd.2)
lines(mean.fd(data.fd.2), lwd=3)

plot.fd(data.fd.3)
lines(mean.fd(data.fd.3), lwd=3)

# Covariance
eval.1 = eval.fd(time, data.fd.1)
image.plot(cov(t(eval.1)))

eval.2 = eval.fd(time, data.fd.2)
image.plot(cov(t(eval.2)))

eval.3 = eval.fd(time, data.fd.3)
image.plot(cov(t(eval.3)))

###----------------------------------------------------------------------------------
### Part III: PCA
### Remember to set ylim!
###----------------------------------------------------------------------------------
pca.data = pca.fd(data.fd.1, nharm=5, centerfns=T)      # nharm = number of PC's

# PCA compute all the pc's, but only n-1 are not null
```

```r
plot(pca.data$values, xlab='j', ylab='Eigenvalues')
plot(pca.data$values[1:n], xlab='j', ylab='Eigenvalues')

plot(cumsum(pca.data$values)[1:n]/sum(pca.data$values), xlab='j', ylab='CPV')

# Explained variance
pca.data$varprop

# First PC
plot(pca.data$harmonics[1,], col=1, ylab='FPC1', ylim=c(-0.1,0.08))

# Second PC
plot(pca.data$harmonics[2,], col=2, ylab='FPC2', ylim=c(-0.1,0.08))

# Plot of FPCs as perturbation of the mean
par(mfrow=c(1,2))
plot.pca.fd(pca.data)

# Scatterplot of the scores
par(mfrow=c(1,2))
plot(pca.data$scores[,1], pca.data$scores[,2], xlab="Scores FPC1",ylab="Scores FPC2",lwd=2)
points(pca.data$scores[n,1], pca.data$scores[n,2],col=2, lwd=4)
plot(pca.data$scores[,1], pca.data$scores[,2],type="n",xlab="Scores FPC1",
     ylab="Scores FPC2")
text(pca.data$scores[,1], pca.data$scores[,2], dimnames(data)[[2]], cex=1)
dev.off()

# Outliers?
head(data)
matplot(eval.1, type='l')
lines(eval.1[,35], lwd=4, col=2)

# --------------------------------
# Scores with 3 FPCs
# (commented points if the 12-th is an outlier)
layout(cbind(1,2,3))
pca_L = pca.data
plot(pca_L$scores[,1],pca_L$scores[,2],xlab="Scores FPC1",ylab="Scores FPC2",lwd=2)
# points(pca_L$scores[12,1],pca_L$scores[12,2],col=2, lwd=4)
plot(pca_L$scores[,1],pca_L$scores[,3],xlab="Scores FPC1",ylab="Scores FPC3",lwd=2)
# points(pca_L$scores[12,1],pca_L$scores[12,3],col=2, lwd=4)
plot(pca_L$scores[,2],pca_L$scores[,3],xlab="Scores FPC2",ylab="Scores FPC3",lwd=2)
# points(pca_L$scores[12,2],pca_L$scores[12,3],col=2, lwd=4)
# --------------------------------
```

```
# File watertemp.txt contains the mean daily water temperature registered at
# 132 monitoring stations in the Adriatic Sea, during the 365 days of 2017.
# The dataset also report the zone of the measurement (Deep, Medium or Surface water).

# ------------------------------------------------------------------------------------
# a) Perform a smoothing of the data through a projection over a Fourier basis
# with 45 basis elements. Report the first 3 Fourier coefficients obtained at the
# Stations 1 and 2.

data = read.table('watertemp.txt', header=T)
data = t(data[,-366])
dim(data)

library(fda)
basis.1    = create.fourier.basis(rangeval=c(0,365), nbasis=45)
time       = 1:365
data.fd.1 = Data2fd(y = data, argvals=time, basisobj = basis.1)
plot.fd(data.fd.1)

as.numeric(data.fd.1$coefs[1:3, 'Station1'])  # 295.33, 33.17, -35.75
as.numeric(data.fd.1$coefs[1:3, 'Station2'])  # 272.04, 27.72, -33,38

# ------------------------------------------------------------------------------------
# b) Perform a functional principal component analysis of the smoothed data obtained
# at point (a). Report the variance explained along the first 5 functional principal
# components, a qualitative plot of the first 3 eigenfunctions
# and the screeplot. Interpret the principal components.

# PCA
pca_data = pca.fd(data.fd.1, n=5, centerfns=TRUE)

# scree plot
plot(pca_data$values)

# variance explained along the first 5 FPC
pca_data$varprop      # PC1: 0.8527091609
                      # PC2: 0.1273200916
                      # PC3: 0.0127698676
                      # PC4: 0.0016227136
                      # PC5: 0.0006766155

# first 3 eigenfunctions
par(mfrow=c(1,3))
plot(pca_data$harmonics[1,], ylab='FPC1', ylim=c(-0.1,0.1))
plot(pca_data$harmonics[2,], ylab='FPC2', ylim=c(-0.1,0.1))
plot(pca_data$harmonics[3,], ylab='FPC3', ylim=c(-0.1,0.1))

# interpret the PC's
par(mfrow=c(1,3))
plot(pca_data)

# 1st: zone calde vs. zone fredde
# 2nd: zone in cui il caldo arriva prima vs. zone in cui arriva dopo
# 3rd: zone in cui, al massimo del caldo (e del freddo) fa più caldo rispetto
#      alla media vs. fa più freddo

# ------------------------------------------------------------------------------------
# c) Having reported a qualitative plot of the scores along the first 2 functional
# principal components, use the categorical variable zone to further enhance
# the interpretations.

data   = read.table('watertemp.txt', header=T)
levels = as.factor(data$Zone)

scores_and_levels = as.data.frame(pca_data$scores[,1:2])
scores_and_levels[, 'Level'] = levels

ind_deep = which(scores_and_levels$Level == 'Deep')
ind_med  = which(scores_and_levels$Level == 'Medium')
ind_surf = which(scores_and_levels$Level == 'Surface')

par(mfrow=c(1,1))
plot(scores_and_levels[,1:2])
points(scores_and_levels[ind_deep, 1:2], col='red', , pch=19)
points(scores_and_levels[ind_med,  1:2], col='blue' , pch=19)
points(scores_and_levels[ind_surf, 1:2], col='green', pch=19)

# Sembrano esserci 3 chiari clusters basati sulla zona

# ------------------------------------------------------------------------------------
# d) Propose a possible dimensionality reduction for the data and discuss the results.

pca_data$varprop

# Basandoci sulla varianza captured dalle prime PC possiamo dire che ridurre la
# dimensione a 2 principal components sarà sufficiente
```